GOETHE UNIVERSITÄT FRANKFURT AM MAIN

IKF Institut für Kernphysik Frankfurt

Masterarbeit

Entwicklung eines Testprogramms für den Auslese-ASIC des CBM-TRD Detektors

Dennis Spicker

Januar 2023

Institut für Kernphysik
am Fachbereich Physik
der Goethe-Universität Frankfurt am Main

Erstgutachter	Prof. Dr. Christoph Blume
Zweitgutachter	Dr. Jan Michel

Inhaltsverzeichnis

1. Einleitung **1**

2. Grundlegende Betrachtungen **2**
2.1. Das Quark-Gluon-Plasma 3

3. Das Compressed Baryonic Matter Experiment **5**
3.1. Der Transition Radiation Detector 7
3.2. Der SPADIC Ausleseprozessor 9
3.2.1. Die Hit-Erkennungs-Logik 11
3.2.2. Die Neighbor-Trigger-Logik 12
3.3. Datenübertragung zwischen SPADIC und Computer 13
3.3.1. Die IPbus Software 14

4. Die SPADIC-Chiptest Software **16**
4.1. Qualitätskontrolle der Ausleseelektronik 16
4.2. Software Design 18
4.3. Das Spadic-Chiptest Programm 21

5. Zusammenfassung **30**
5.1. Ausblick 30

6. Verzeichnisse **31**

A. SPADIC Konfiguration **35**
A.1. Einstellungen des Analogteils 35
A.2. Einstellungen des Digitalteils 37
A.3. Die Neighbor Trigger Matrix 41

B. Spadic-Chiptest Zusatzmaterial **43**

1. Einleitung

Diese Arbeit beschäftigt sich mit hardwarenaher Programmierung im Umfeld eines Experiments der Hochenergiephysik.

Das Compressed Baryonic Matter (CBM) Experiment an der Facility for Antiproton and Ion Research (FAIR), welche zurzeit beide am GSI - Helmholtzzentrum für Schwerionenforschung (GSI) im Bau sind, soll in wenigen Jahren neue Erkenntnisse im Bereich der hochenergetischen Schwerionenphysik bringen. Ein Teil des Experiments wird der Transition Radiation Detector (TRD) sein, welcher mit einer komplexen, eigens entwickelten Elektronik ausgestattet sein wird, um Messdaten digital aufzunehmen und an einen Datenspeicher weiterzuleiten. Herausforderungen für die Elektronik bestehen unter anderem in den großen Datenmengen, die verarbeitet werden müssen und der hohen Strahlenbelastung, der die Elektronik direkt am Detektor ausgesetzt sein wird.

Während der Produktion der Detektormodule des TRD soll eine Qualitätskontrolle der Elektronik stattfinden. Hierfür wird eine Software benötigt, die definierte Tests automatisch ausführt und die Resultate am Bildschirm anzeigen sowie für die spätere Referenzierung abspeichern kann. Für die vorliegende Masterarbeit wurde diese Software konzipiert und anschließend programmiert.

Bevor in Kapitel 4 die Einzelheiten zu diesem Programmierprojekt dargestellt werden, gibt die Arbeit zunächst einen Überblick über die physikalischen Hintergründe und stellt dann das CBM Experiment sowie insbesondere den TRD und die Ausleseelektronik detailliert vor.

2. Grundlegende Betrachtungen

Dieses Kapitel umreißt kurz die theoretischen Aspekte, welche für die wissenschaftliche Arbeit in der Hochenergiephysik relevant sind.

Die Physik hat sich zur Aufgabe gemacht, die grundlegenden Bausteine und Gesetzmäßigkeiten des Universums zu ergründen. Bei näherer Betrachtung ergeben sich eine ganze Reihe grundsätzlicher Fragen: Aus welchen Bausteinen ist Materie aufgebaut? Was sind die möglichen Zustände von Materie? Wie wechselwirken die Bausteine miteinander?

Mit dem aktuellen Stand der Forschung können diese Fragen schon sehr detailliert beantwortet werden, auch wenn eine einheitliche „Theorie von allem“, die alle Phänomene gleichermaßen beschreiben kann, noch fehlt.

Im Bereich der Hochenergiephysik liegt das Hauptaugenmerk auf den kleinsten, unteilbaren Grundbausteinen der Materie von Atomkernen und ihrer Wechselwirkung miteinander. Dies sind im Speziellen die Quarks und Gluonen, welche durch die starke Wechselwirkung in Verbindung miteinander treten.

Bei der Erforschung von Atomkernen wollten Wissenschaftler seit den 1960er Jahren herausfinden, aus welchen Grundbausteinen Protonen und Neutronen zusammengesetzt sind. Dafür versuchten sie in Beschleunigerexperimenten immer mehr Energie in das zu messende System einzubringen, wodurch die zugrunde liegenden Teilchenzustände messbar gemacht werden sollten. Dabei fiel auf, dass mit steigender Energie die Zahl der neuen Teilchen exponentiell anstieg. Es wurde schnell klar, dass es sich dabei aber nicht um die gesuchten Grundbausteine selbst handelte, sondern um neue Kombinationen der Grundbausteine.

Rolf Hagedorn entwickelte anhand dieser Beobachtung 1968 das sogenannte „Statistical Bootstrap Modell“. Das Erhöhen der Energie des Systems führt irgendwann nicht mehr zur Erhöhung der Temperatur, sondern zur Erzeugung neuer Teilchenzustände. Hagedorn schloss daraus, dass es eine obere Temperaturgrenze für Kernmaterie geben müsse, die nicht überschritten werden kann. Aus der gemessenen Teilchenstatistik bestimmte er diese kritische Temperatur zu $T_c \simeq 150 - 200$ MeV [1].

2.1. Das Quark-Gluon-Plasma

Was passiert wenn T_c überschritten wird, beschreibt die Theorie der Quantenchromodynamik (QCD), die ab Mitte der 1970er Jahre entwickelt wurde. Dabei war es zunächst wichtig, zu erkennen, dass die Kopplungskonstante der starken Wechselwirkung – im Gegensatz zur elektromagnetischen Wechselwirkung – mit steigendem Impulsübertrag abnimmt. Wenn Kernmaterie aufgeheizt und/oder komprimiert wurde, also große Impulsüberträge zwischen den Teilchen vorherrschen, so wird die Wechselwirkung zwischen den Quarks asymptotisch klein, sie können sich aus den Hadronen, in denen sie zuvor gebunden waren, lösen und sich quasi-frei bewegen. Dies wird als Deconfinement bezeichnet [1].

In der Quantenelektrodynamik (QED) sind theoretische Berechnungen aufgrund der konstant kleinen Kopplungskonstante $\alpha_e = 1/137$ durch störungstheoretische Methoden lösbar. Dies lässt sich in die QCD nur für sehr große Impulsüberträge – und damit ebenfalls kleine Kopplungskonstante α_s – übertragen. Bei kleinerem Impulsübertrag (sogenannten „weichen“ Kollisionen) verschwindet der Betrag der Kopplungskonstante nicht. Auch die Selbstwechselwirkung der Gluonen, die als Austauschteilchen der starken Wechselwirkung selbst auch Farbladung tragen, muss berücksichtigt werden. Numerische Lösungsverfahren sind daher nur dann anwendbar, wenn die kontinuierliche Raumzeit in ein diskretes Gitter (englisch „Lattice“) überführt wird. Dieser Ansatz wird Gittereichtheorie, oder auch Lattice-QCD (LQCD) genannt [2].

Der Übergang von in Hadronen gebundenen zu freien Quarks ist als physikalischer Phasenübergang charakterisierbar. Es handelt sich daher um einen bisher unbekannten Materiezustand, für den sich der Begriff Quark-Gluon-Plasma (QGP) etabliert hat. Abbildung 2.1 zeigt ein Phasendiagramm hadronischer Materie, wie Physiker es sich nach aktuellem Kenntnisstand vorstellen. Der Bereich, in welchem der Phasenübergang stattfindet, ist dort als hellrote Fläche gekennzeichnet. In normalen, schweren Atomkernen liegt die Nukleonendichte bei $n_0 = 0.17$ fm^{-3}, dies entspricht einer Energiedichte von $\epsilon_0 \approx 0.16$ GeV fm^{-3} und einem baryochemischen Potential $\mu_B \approx 800$ MeV. Der Übergang zum QGP wird bei ungefähr $n_c = 0.56$ fm^{-3} beziehungsweise $\epsilon_c \approx 1 - 2$ GeV fm^{-3} erwartet. Für $T \to 0$ entspricht dies ungefähr $\mu_B^c \approx 1300$ MeV, während sich für $\mu_B \to 0$ aus LQCD-Rechnungen ergibt, dass $T_c \approx 156$ MeV ist. Bei diesen Werten befindet sich genug Energie im System, um das Confinement aufzuheben [1].

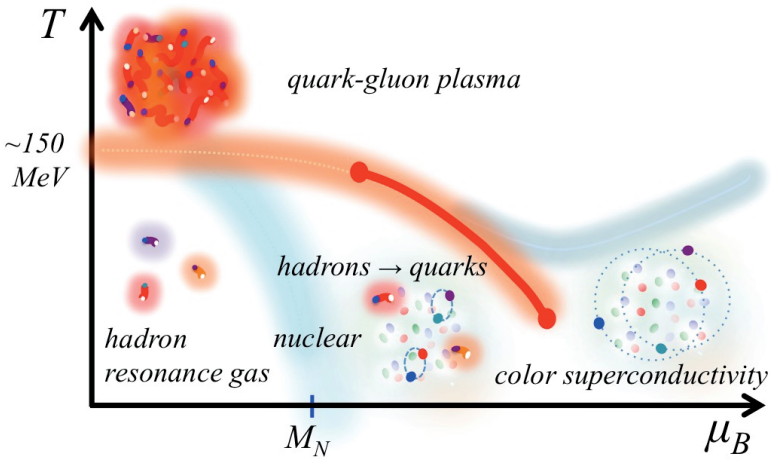

Abbildung 2.1.: Schematisches Phasendiagramm dichter Kernmaterie mit der Temperatur T und dem baryochemischen Potenzial μ_B als Maß der Netto-Baryonendichte im System. Bei $T \to 0$ gibt es Nukleonen nur oberhalb von $M_N \approx 940$ MeV [3].

Von näherem Interesse sind nun mehrere Aspekte; unter anderem die Ordnung des Phasenübergangs, der Wert der kritischen Temperatur, Druck und Energiedichte des Mediums. Neuere LQCD Rechnungen, die endliche Quarkmassen einbeziehen, kommen zum Beispiel zu dem Ergebnis, dass der Phasenübergang bei kleinen Werten der Netto-Baryonendichte μ_B zunächst kontinuierlich (cross-over) stattfindet. Mit steigender Netto-Baryonendichte wird erwartet, dass ein kritischer Punkt und dann ein Phasenübergang erster Ordnung, in Abbildung 2.1 als durchgezogene, rote Linie eingezeichnet, erreicht wird.

Die von Hagedorn statistisch ermittelte Temperatur liegt erstaunlich nah an der Grenztemperatur des Phasenübergangs bei verschwindendem baryochemischem Potenzial $\mu_B \to 0$.

3. Das Compressed Baryonic Matter Experiment

Das Compressed Baryonic Matter (CBM) Experiment will die experimentelle Erforschung des QGP weiter voranbringen und Bereiche des QCD - Phasendiagramms vermessen, die mit bisherigen Experimenten erst wenig erforscht wurden. Der angestrebte Bereich liegt bei mittleren Temperaturen und $\mu_B > 0$. Dort sagt die Theorie einen Phasenübergang voraus, der in einem kritischen Punkt endet. Ob dies tatsächlich so ist, soll experimentell nachgewiesen werden.

Um die dafür nötige hohe Netto-Baryonendichte im Labor zu erreichen, wird am GSI - Helmholtzzentrum für Schwerionenforschung (GSI) in Darmstadt aktuell eine neue Beschleunigeranlage namens Facility for Antiproton and Ion Research (FAIR) gebaut. Hierdurch werden die bestehenden Schwerionen-Beschleuniger um das Schwer-Ionen Synchrotron (SIS)-100 erweitert. Damit wird es möglich sein, Protonen auf bis zu 29 GeV, Kerne mit $Z/A = 0.5$ auf 14 Gev pro Nukleon (AGeV) und zum Beispiel Gold-Ionen auf 11 AGeV zu beschleunigen. Das als Fixed-Target Experiment aufgebaute CBM-Experiment wird an das SIS-100 angeschlossen sein und kann dann mit bis zu 10^9 Gold-Ionen pro Sekunde versorgt werden.

Die Untersuchung der Schwerionenkollision stützt sich auf viele verschiedene Observablen, die auf unterschiedliche Arten gemessen werden. Essentiell ist dabei immer die Flugbahnen der entstandenen Teilchen nachzuvollziehen, ihre Geschwindigkeit und ihren Impuls zu messen. Anhand dieser Werte können Teilchen identifiziert werden und sie können bestimmten Reaktionen zugeordnet werden. Erst das Zusammenspiel verschiedener Detektor-Komponenten macht die Messung all dieser Größen möglich. Für die Impulsbestimmung, zum Beispiel, ist es nötig, die Teilchentrajektorie in einem Magnetfeld zu bestimmen. Das CBM-Experiment wird aus den folgenden Komponenten bestehen, deren Anordnung in Abbildung 3.1 dargestellt ist [4].

Der supraleitende Magnet erzeugt ein homogenes Magnetfeld mit einer Magnetfeldstärke von 1T. Er hat eine große Öffnung, um die Tracking-Detektoren aufzunehmen und eine möglichst große Akzeptanz zu gewährleisten.

Der Micro Vertex Detector (MVD) befindet sich im Magneten, sehr nah am Target. Er bietet eine sehr genaue Spurrekonstruktion, um anhand von Sekundärvertizes – unter anderem – Open-Charm Teilchen zu identifizieren.

Das Silicon Tracking System (STS) befindet sich ebenfalls im Magneten, direkt hinter dem MVD. Er verwendet Halbleiterdetektoren für Tracking und Impulsmessung aller geladenen Teilchen.

Der Muon Chamber (MUCH) Detektor besteht aus mehreren Lagen von Hadronen-Absorbern (Kohlenstoff- und Stahlplatten) mit dazwischen liegenden Gas Electron Multiplier (GEM)-Detektoren, wodurch eine impulsabhängige Myonen-Identifikation möglich ist. Er wird in Strahlrichtung direkt nach dem Magneten aufgebaut.

Der Ring Imaging Cherenkov Detector (RICH) kann statt des MUCH Detektors in das Setup eingefügt werden. Er identifiziert Elektronen anhand ihrer Cherenkov-Strahlung.

Der Transition Radiation Detector (TRD) steht hinter dem MUCH oder dem RICH Detektor. Er wird in Kapitel 3.1 detailliert beschrieben.

Der Time Of Flight (TOF) Detektor befindet sich hinter dem TRD. Er misst die Flugzeit der Teilchen mit sehr hoher Auflösung und ermöglicht damit die Identifikation der verschiedenen Hadronen anhand ihrer Geschwindigkeit.

Der Projectile Spectator Detector (PSD) macht die Bestimmung der Zentralität und Lage der Reaktionsebene möglich, indem gemessen wird, welcher Anteil an Nukleonen nicht an der Kollision mit dem Target teilgenommen hat.

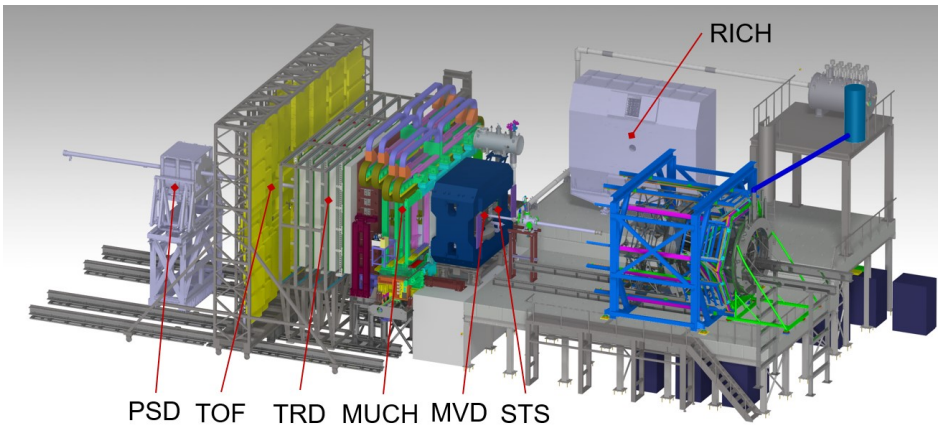

Abbildung 3.1.: Geplanter Aufbau des CBM-Experiments im Cave. Hier im Myonen-Setup, der RICH-Detektor steht auf der Parkposition. Auf der rechten Seite des Bildes ist auch das HADES-Experiment abgebildet. Der Teilchenstrahl kommt von rechts, der Beam-Dump befindet sich links. [5]

3.1. Der Transition Radiation Detector

Das Design des TRD ist darauf ausgelegt vor allem Elektron-Positron-Paare (Dielektronen) zu identifizieren. Diese sind eine sehr wichtige Observable in der Hochenergiephysik, da vor allem leichte Vektormesonen (ω, ρ, ϕ) und Quarkonia über diesen Kanal zerfallen. Die Elektronen wechselwirken nicht über die starke Wechselwirkung und können so den Feuerball der Kollision unbeeinflusst verlassen. Da bei den Teilchenkollisionen aber auch sehr viele Pionen entstehen, die ein identisches Ladungs-Impuls-Verhältnis wie die Elektronen haben können, wird ein Detektor benötigt, der Elektronen und Pionen unterscheiden kann.

Dies kann der TRD leisten. Er wird aus Multi Wire Proportional Chamber (MWPC) Elementen bestehen, die mit einem Radiator ausgestattet sind. Die Elektronen erzeugen in diesen Radiatoren Übergangsstrahlung (Transition Radiation), während die schwereren und damit langsameren Pionen dies nicht tun. Es ist zudem möglich, den spezifischen Energieverlust $\langle dE/dx \rangle$ eines Teilchens im Detektor zu bestimmen, wodurch eine Identifikation von Kern-

fragmenten durchgeführt werden kann. Der Aufbau und das Funktionsprinzip des Detektors wird durch Abbildung 3.2 verdeutlicht. Die Dicke des Gasvolumens in Strahlrichtung (z-Richtung) ist aufgeteilt in 5 mm Driftbereich und $3.5 + 3.5$ mm Verstärkungsbereich.

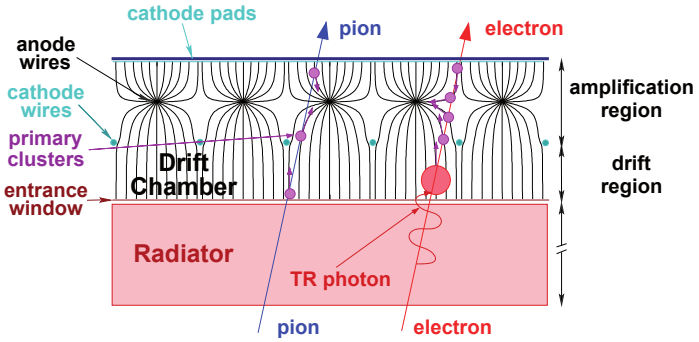

Abbildung 3.2.: Schematischer Schnitt durch eine TRD Kammer der das Arbeitsprinzip zeigt. Die schwarzen Linien zeigen den Verlauf der elektrischen Feldlinien. [4]

Da die hintere Kathodenebene, die auch die Rückwand des Detektors bildet (Padplane), in kleine, rechteckige Pads unterteilt ist, welche einzeln von der Detektorelektronik ausgelesen werden, kann auch eine Ortsinformation abgeleitet werden. Die Ortsauflösung entlang der kurzen Padseite wird circa 300 μm betragen, entlang der langen Padseite wird sie in der Größenordnung der Seitenlänge der Pads liegen. Der Detektor wird aus vier identischen Lagen bestehen, bei denen in jeder zweiten Lage die Detektorkammern um 90 Grad rotiert sein werden, um die hohe Ortsauflösung in x- und y-Richtung zu nutzen. Abbildung 3.3 zeigt einen Ausschnitt einer Padplane vor der Montage der Detektorkammer.

Ein geladenes Teilchen, das den Detektor durchfliegt, wird das Gas in der MWPC ionisieren. Die entstehenden Elektronen und Ionen werden zu den Anoden- beziehungsweise Kathodendrähten driften und dabei eine Ladung auf der Padplane influenzieren. Diese Ladung wird von der Front-End-Elektronik (FEE) gemessen. Sie ist proportional zum Energieverlust des Teilchens im Detektor. Die Funktionsweise der Signalauslese wird in Kapitel 3.2 genauer beschrieben.

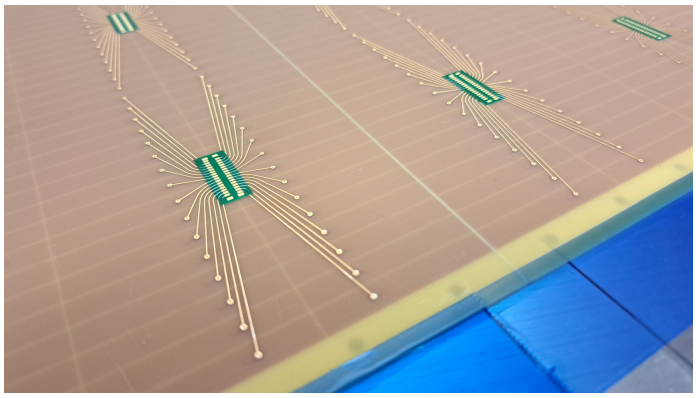

Abbildung 3.3.: Die Padplane - Leiterplatte, Fotografiert von der Rückseite der Kammer. Es sind die Leiterbahnen und Kontakte zu sehen, welche die einzelnen Pads elektrisch mit der Ausleseelektronik verbinden werden. Durchscheinend sind auch die einzelnen Pads auf der Innenseite zu erkennen, hier in der Größe $0.67 \cdot 4.0$ cm^2. [6]

3.2. Der SPADIC Ausleseprozessor

Zur Auslese der Detektorsignale an der Padplane wird in Zusammenarbeit mit der Universität Heidelberg ein Application-Specific Integrated Circuit (ASIC) entwickelt, der Self-triggered Pulse Amplification and Digitization asIC (SPADIC). Er hat 32 Eingangskanäle, mit denen je ein Pad ausgelesen werden kann. Ein Signal vom Detektor durchläuft im Chip zuerst einen ladungssensitiven Verstärker, kombiniert mit einem Signal-Shaper[1]. Dessen Hauptaufgabe ist die Verbreiterung des Signals, was die Digitalisierung erleichtert. Dies erfolgt anschließend in einem Analog to Digital Converter (ADC). Darauf folgt ein digitaler Filter[1] und die Hit-Erkennungs-Logik. Letztere entscheidet, ob das gemessene Signal für die weitere Auslese interessant ist und übergibt die Daten dann an den Message-Builder, der daraus eine Hit-Message erstellt und diese dann über die serielle Verbindung (e-Link) an die nächste Stufe der Datenerfassung, auch Data Acquisition (DAQ) genannt, weiterleitet. Der Weg des Signals kann anhand der schematischen Darstellung der SPADIC-Komponenten

[1] Mehr Informationen zu Shaper und Filter in [4, Kapitel 7.1]

in Abbildung 3.4 nachvollzogen werden. Wenn die Hit-Erkennungs-Logik veranlasst, eine Hit-Message zu erzeugen, wird dies „Trigger“ genannt.

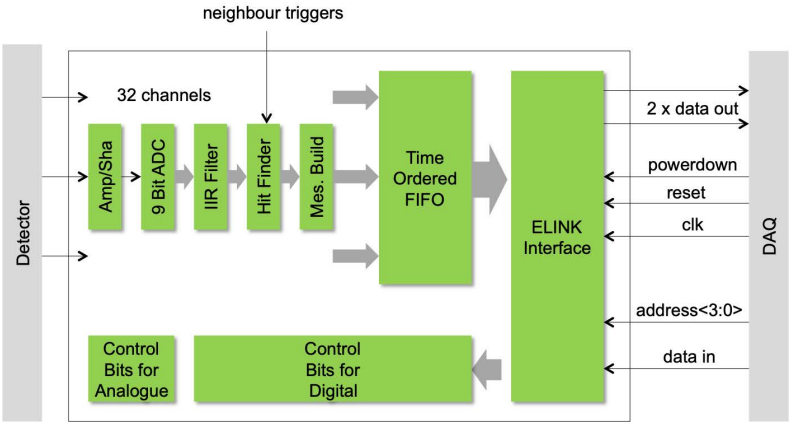

Abbildung 3.4.: Blockdiagramm des SPADIC Chips [7].

Die 32 Auslesekanäle sind intern im Chip in zwei Gruppen mit je 16 Kanälen aufgeteilt, jede Gruppe hat einen eigenen Uplink[2] Kanal. Dazu kommt noch ein Downlink[3] pro Chip, über den das globale Zeit- / Taktsignal verteilt wird, die Uplinks synchronisiert werden und Einstellungs-Befehle an den SPADIC gesendet werden können. Eine Hit-Message, die vom SPADIC über den e-Link weiter gesendet wird, beinhaltet die folgenden Informationen:

- Kanalnummer
- Zeitstempel
- Multihit Kennzeichnung. Ist ein neues Ereignis aufgetreten, bevor das vorherige komplett aufgezeichnet wurde?
- Hit-Typ. Es gibt vier Möglichkeiten: Externer Trigger, Selbst-Trigger, Nachbar-Trigger, gleichzeitiger Selbst- und Nachbar-Trigger
- Bis zu 32 ADC-Werte, die den zeitlichen Verlauf des Signals darstellen.

[2]Datenverbindung von der FEE in Richtung DAQ
[3]Datenverbindung von der DAQ in Richtung FEE

Die Platinen, auf denen sich die SPADIC Chips befinden, werden Front-End-Board (FEB) genannt. Sie enthalten alle Komponenten, die für den Betrieb der ASICs nötig sind. Dies sind im Wesentlichen die Spannungsversorgung, der Anschluss an den Detektor und die Kabelverbindung für den e-Link. Es wird FEBs mit zwei oder drei SPADICs geben, je nach Pad-Dichte in den Detektorkammern. In der aktuellen Entwicklungsphase stehen Platinen mit je einem Chip zur Verfügung (Single-FEB).

Die zuvor beschriebenen Funktionen des SPADIC lassen sich anhand einer Vielzahl von Einstellungen konfigurieren, um bei allen Messungen ein bestmögliches Ergebnis zu erzielen. Hierfür gibt es im SPADIC zwei Schieberegister, deren Dateninhalt das Verhalten des Chips steuert. Ein Register beinhaltet dabei die Konfiguration des analogen Teils, also des Verstärkers und des Shapers. Das andere Register beinhaltet die Konfiguration des digitalen Teils, unter anderem also des ADC und der Hit-Erkennungs-Logik. Die Register sind in Abbildung 3.4 ganz unten dargestellt.

3.2.1. Die Hit-Erkennungs-Logik

Im Betrieb misst der SPADIC kontinuierlich das Ladungssignal an jedem angeschlossenen Pad und entscheidet selbst, ob das Signal einem Hit im Detektor entspricht und deshalb weitergegeben wird, oder nicht. Diese Entscheidung wird von der Hit-Erkennungs-Logik gefällt, die zwei unterschiedliche Betriebsmodi hat: Den „klassischen Modus“ und den neu entwickelten, „flexiblen Modus“. Hier wird der klassische Modus beschrieben, da dieser in bisherigen Tests und bei Strahlzeiten verwendet wurde. Um zwischen den beiden Modi zu wechseln, muss ein Bit im Register für den Digitalteil gesetzt werden: Wenn `REG_useOldHitDet` $= 1$ ist, dann wird der klassische Modus verwendet (siehe auch Tabelle in Anhang A.2).

Der klassische Modus kennt wiederum zwei Betriebsarten: Absolut und differenziell. Worin diese sich unterscheiden wird deutlich, wenn die Funktion der Hit-Erkennungs-Logik im Folgenden näher beschrieben wird. Um die Betriebsart zu wählen, wird im Register des Digitalteils das Bit `REG_compDiffMode` entsprechend gesetzt. Es gibt zwei Trigger-Schwellen, englisch thresholds, die der Benutzer für jeden Kanal einstellen kann. Das Signal wird zu bestimmten Zeitpunkten mit den thresholds `TH1` und `TH2` verglichen und wenn es sie übersteigt, wird das Signal ausgelesen und weiterverarbeitet. Die Triggerlogik

wird durch Gleichung (3.1) vollständig beschrieben.

$$x(t) = \begin{cases} y(t) & \text{if } \texttt{REG\_compDiffMode} = 0 \\ y(t) - y(t-1) & \text{if } \texttt{REG\_compDiffMode} = 1 \end{cases} \tag{3.1a}$$

$$h(t) = (x(t-1) > \texttt{TH1}) \ \wedge \ (x(t) > \texttt{TH2}) \tag{3.1b}$$

$$\text{hit}(t) = h(t) \ \wedge \ \neg h(t-1) \tag{3.1c}$$

Hier ist $y(t)$ das gemessene, digitale Signal zu einem Zeitpunkt t. Und $x(t)$ ist, abhängig von der Betriebsart, der von der Logik verarbeitete Wert des Signals. In Gleichung (3.1a) wird deutlich, dass die differenzielle Betriebsart die Differenz zwischen zwei aufeinanderfolgenden Messwerten betrachtet. Ein Hit wird ausgegeben, wenn $\text{hit}(t) = 1$, beziehungsweise *wahr* ist. Es wird ersichtlich, dass für diese Logik vier aufeinanderfolgende Messwerte $y(t), \ \ldots, \ y(t-3)$ gespeichert werden müssen. Anders formuliert wird ein Hit erkannt, wenn $x(t-2) < \texttt{TH1} < x(t-1)$ und gleichzeitig $x(t-1) < \texttt{TH2} < x(t)$ ist.

3.2.2. Die Neighbor-Trigger-Logik

Ein wichtiges Feature des SPADIC ist die „Neighbor-Trigger-Logik“, die es erlaubt, Signale von benachbarten Pads zu speichern, wenn ein Hit detektiert wurde. Dies ist nötig, da die Ladungswolke eines Teilchens über mehrere Pads ausgedehnt ist, aber in nur einem Pad die Bedingungen für einen Hit erfüllt sind. Um die volle Information des Hits zu erhalten, werden aber die Daten von allen beteiligten Pads benötigt. Dies funktioniert auch über die Grenzen eines einzelnen SPADIC hinaus.

Diese Technik sorgt auch für die gute Ortsauflösung des Detektors, da anhand einer Schwerpunktberechnung aus mehreren Hit-Messages eine genauere Ortsinformation vorliegt. Die Neighbor-Trigger-Logik wird im Experiment so konfiguriert sein, dass Trigger nur in Richtung der kurzen Padseiten weitergegeben werden, was die hohe Ortsauflösung auf diese Koordinate beschränkt.

Abbildung 3.5 zeigt schematisch zwei Padreihen im Detektor und beispielhaft einen Hit als hellrotes Kreuz markiert. Der SPADIC-Kanal, der an dem getroffenen Pad angeschlossen ist (hier Kanal 10) registriert einen Hit und hat sich somit selbst getriggert (Self Trigger). Um die gesamte Ladung zu messen, wird der SPADIC nun so konfiguriert, dass auch die beiden benachbarten Kanäle (hier 7 und 8) ausgelöst werden (Neighbor Trigger). Dies kann

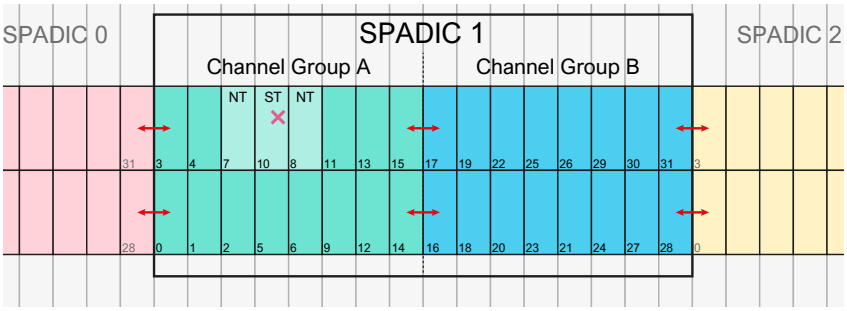

Abbildung 3.5.: Schematische Darstellung eines Padplane-Bereichs, der von einem SPADIC ausgelesen wird. Die Roten Pfeile markieren Stellen, an denen Neighbor-Trigger Signale zur nächsten Kanalgruppe übertragen werden müssen. Die Zahlen in den Pads bezeichnen den jeweils verbundenen Kanal des SPADIC.

mit genauso großer Wahrscheinlichkeit auch an der Grenze zwischen zwei Kanalgruppen oder an der Grenze zum benachbarten SPADIC-Chip passieren. Ersteres wäre zum Beispiel ein Hit in dem Pad an Kanal Nummer 15, letzteres würde bei einem Hit im Pad mit Kanal Nummer 3 eintreten. Hier müssen jeweils Kanäle getriggert werden, die nicht in derselben Kanalgruppe liegen. Dafür besitzt jede Kanalgruppe zwei bidirektionale „Trigger-Busse“ mit jeweils vier Kanälen, welche die Gruppe mit ihren zwei benachbarten Gruppen verbinden. Dabei ist nicht relevant, ob die Nachbargruppe im selben Chip liegt oder nicht. Anhang A.3 bietet weitere Informationen zur Konfiguration der Neighbor-Trigger-Logik.

3.3. Datenübertragung zwischen SPADIC und Computer

Im finalen Aufbau des CBM Experiments werden die e-Links von der FEE zu ReadOut Boards (ROBs) geführt. Diese sind mit einem oder mehreren GigaBit Transceiver ASIC (GBTx) ausgestattet, welche die Daten anhand eines eigenen Übertragungsprotokolls (GBT-link) über optische Datenverbindungen zum Common Readout Interface (CRI) führen, welches sich in einem Serverraum nahe des Experiment-Cave befindet [8].

Für Tests im Labor, sowie für die Entwicklung von Firmware und Software existiert ein Prototyp-Setup, das die Steuerung und Auslese des SPADIC über das IPbus[4] Kommunikationsprotokoll ermöglicht. Mit diesem Setup kann zurzeit ein SPADIC v2.2 Single-FEB durch ein AMC FMC Carrier Kinter (AFCK) Board angesteuert werden. Hauptbestandteil des AFCK ist ein Field-Programmable Gate Array (FPGA).

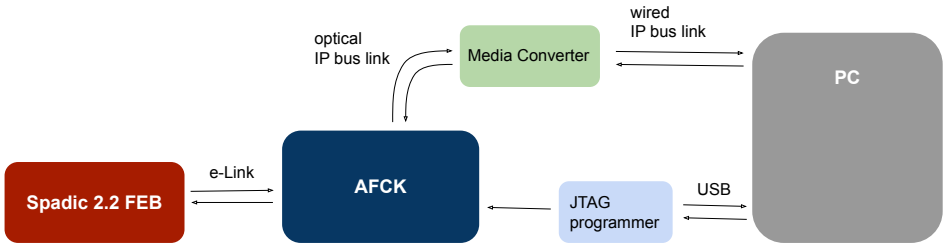

Abbildung 3.6.: Schematische Darstellung der Prototyp-Auslesekette (DAQ-Chain) im Labor.

Abbildung 3.6 zeigt schematisch die Verbindungen zwischen den Komponenten. Der FPGA des AFCK-Boards kann über den JTAG-Programmer mit der passenden Firmware programmiert werden, die als Bit-File vorliegt. Der e-Link zwischen SPADIC-FEB und AFCK erfolgt mit einem 40-poligen Twisted-Pair-Flachbandkabel. Der IPbus-Link zwischen AFCK-Board und Computer verwendet ein Ethernet Übertragungsprotokoll. Er ist AFCK-seitig als optischer Glasfaser-Anschluss und PC-seitig als RJ45 Verbindung ausgeführt. Daher ist ein entsprechender Media-Converter zwischengeschaltet.

Die Funktionsweise dieses Aufbaus wird in [9, Kap. 1.3.2] ausführlich beschrieben.

3.3.1. Die IPbus Software

Die IPbus Software wird am Conseil Européen pour la Recherche Nucléaire (CERN) speziell entwickelt, um eine zuverlässige Hochleistungs-Steuerungsverbindung für Teilchenphysik-Elektronik bereitzustellen. Sie implementiert

[4]Siehe Kapitel 3.3.1.

ein einfaches Steuerungsprotokoll für FPGA-basierte Hardware. Die Software besteht aus zwei Hauptkomponenten [10]:

- μHAL (Hardware Access Library). Programmierschnittstelle für Lese- und Schreib-Operationen in Registern von angeschlossenen Geräten. Verfügbar in C++ und Python.
- ControlHub. Software-Anwendung, die den gleichzeitigen Hardware-Zugriff von mehreren μHAL-Clients vermittelt. Wird auf dem DAQ-PC ausgeführt.

Die Clients kommunizieren über das IPbus-Protokoll, das eine Ethernet-Verbindung verwendet. Die verwendeten Komponenten müssen daher eine Ethernet-Schnittstelle bieten.

Weitere Informationen zu IPbus, darunter die Dokumentation und der Quellcode, finden sich unter `https://ipbus.web.cern.ch`[5].

[5] abgerufen am 8.12.2022

4. Die SPADIC-Chiptest Software

In diesem Kapitel geht es um die Aufgabenstellung für diese Masterarbeit und um die entstandenen Resultate. Zuerst werden die Anforderungen an das Projekt dargestellt, dann erfolgt eine detaillierte Beschreibung des entwickelten Programms, inklusive der Diskussion bestimmter Design-Entscheidungen. Dieser Teil der Arbeit soll auch als Benutzerhandbuch für die Software dienen.

4.1. Qualitätskontrolle der Ausleseelektronik

Während der zukünftigen Serienproduktion der Detektorkammern des TRD werden insgesamt über 1000 FEBs mit je zwei oder drei SPADIC-Chips an den Kammern verbaut. Es erscheint sinnvoll, die Elektronik vor dem Einbau zu überprüfen. Dies war der Grundgedanke bei der Formulierung der Aufgabenstellung für diese Masterarbeit.

Aufgrund der großen Anzahl der zu prüfenden Platinen ist es wünschenswert, so viele Arbeitsschritte wie möglich zu automatisieren. Zudem soll es möglich sein, die Prüfung schon nach einer kurzen Einweisung selbstständig und trotzdem mit hoher Qualität durchführen zu können, um unter anderem auch studentische Hilfskräfte mit dieser Arbeit zu betrauen. Zu testen wäre dann zum einen die Hardware, also der korrekte Sitz aller Bauteile auf der Platine, sowie die elektrische Verbindung. Zum anderen soll die Funktion der SPADICs in der Auslesekette überprüft werden. Dazu zählt zum Beispiel, ob grundsätzlich eine Datenverbindung über das zugrundeliegende Kommunikationsprotokoll zustande kommt. Eine entsprechende Apparatur und eine zugehörige Prüfprozedur können im Labor allerdings erst entwickelt werden, wenn das finale Design der Ausleseelektronik feststeht. Da dies aktuell noch nicht der Fall ist, wurde entschieden, zunächst nur die softwareseitigen Grundvoraussetzungen zu schaffen, die ein so umfangreiches Projekt benötigt.

Daher wurden die im Folgenden beschriebenen Randbedingungen für dieses Projekt definiert. Es soll ein Programm entwickelt werden, das

- mit einem SPADIC-FEB mittels Ipbus und AFCK kommuniziert.
- lokal auf einem Computer ausgeführt wird, an dem auch die übrigen Komponenten der DAQ-Chain angeschlossen sind.

- über eine grafische Benutzeroberfläche verfügt, die so gestaltet ist, dass das Programm auch von Nichtexperten bedient werden kann.
- die automatische Ausführung bestimmter, zuvor definierter, Tests ermöglicht.
- ein Protokoll der Tests erstellt und abspeichert.
- Einstellungen des SPADIC anzeigen und verändern kann.
- zu Testzwecken Messdaten auslesen und anzeigen kann.
- eine Grundlage bietet, auf welcher bei der Entwicklung des finalen Testprogramms aufgebaut werden kann.

Das in Kapitel 3.3 beschriebene Protoyp-Setup wurde in einem Laborraum des Institut für Kernphysik Frankfurt (IKF) aufgebaut, um die Software während der Entwicklung testen und debuggen zu können. Dafür wurde ein neuer Desktop-PC mit einer zusätzlichen Netzwerkkarte ausgestattet und mit dem Linux-Betriebssystem Debian Version 10 („Buster“) aufgesetzt. Der zweite Netzwerkanschluss dient der Verbindung mit dem AFCK via IPbus Protokoll. Abbildung 4.1 zeigt die Komponenten auf dem Labortisch.

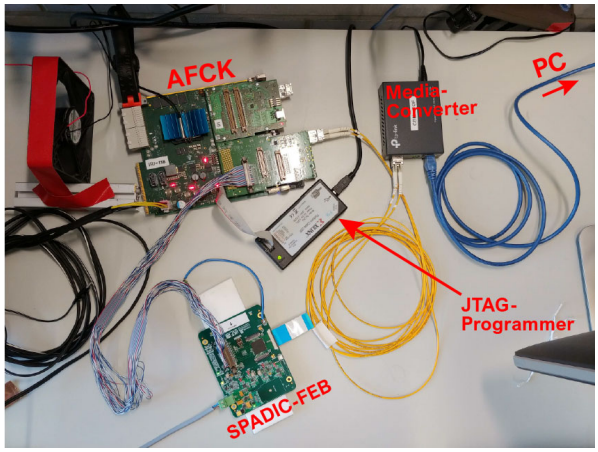

Abbildung 4.1.: Aufbau des Prototyp-Setups im Labor.

Die Firmware des AFCK ist ein wichtiger Teil dieses Aufbaus, sie wurde von David Schmidt[1] im Rahmen seiner Masterarbeit [9] entwickelt. Dabei wurde auch eine bestehende SPADIC-Kontroll-Software überarbeitet und mit einem Command-Line-Interface ausgestattet. Diese Software mit dem Namen „Spadic-Control“ verwendet die μHAL-Bibliothek (siehe auch Kap. 3.3.1), sie wurde in Python Version 2.7 geschrieben und bildet den Startpunkt für die Entwicklung des Test-Programms in dieser Arbeit.

4.2. Software Design

Die neue Software wurde mit Python 3 entwickelt, da so einige Teile der bereits existierenden Software mit wenig Aufwand übernommen werden konnten. Dabei handelt es sich hauptsächlich um diejenigen Teile, die das IPbus Protokoll mit der μHAL-Bibliothek implementieren.

Python 3 bietet für die Erstellung von grafischen Benutzeroberflächen, englisch Graphical User Interface (GUI), verschiedene Möglichkeiten, von denen `tkinter` und `pyQt` die meistverwendeten sind. Die Wahl fiel auf pyQt5, eine Python Portierung des in C++ geschriebenen Frameworks Qt, Version 5. PyQt5 bietet plattformunabhängig ein natives Aussehen, ist sehr flexibel und lässt sich für wachsende Projekte leicht skalieren. Die GUI und der Quellcode des eigentlichen Programms, auch Backend genannt, bleiben hier getrennt, was die Weiterentwicklung erleichtert. Für die nicht-kommerzielle, kostenfreie Nutzung wird pyQt5 unter der GNU General Public License v3 lizenziert.

Mit dem Qt-Framework wird ein Programm in drei Schichten aufgebaut

1. *Das Frontend, die GUI.* Klassen und Funktionen, die das Aussehen der Benutzeroberfläche bestimmen.

2. *Die Controller.* Klassen, die Benutzereingaben am Frontend weitervermitteln an das Backend, indem dort die passenden Funktionen aufgerufen werden.

3. *Das Backend.* Das zugrundeliegende Programm, hier gibt es keine Abhängigkeiten von Qt.

Anhand dieses Schemas wurde nun das geforderte Programm entwickelt. Es erhielt den Arbeitstitel **„Spadic-Chiptest“**. Als Frontend wurde eine GUI

[1] verh. Schledt

entworfen und aus sogenannten Widgets zusammengesetzt. Widgets sind Bausteine aus der Qt-Bibliothek. Ein Widget kann ein einzelner Button sein, aber auch eine ganze Eingabemaske, die aus mehreren Buttons und Eingabefeldern zusammengesetzt ist. Für die grafische Darstellung von Messwerten in der GUI wurde das Python-Paket `matplotlib` verwendet.

Das Backend baut auf dem bereits bestehenden „Spadic-Control“ Programm auf. Es übernimmt dessen Implementierung der μHAL-Bibliothek. Da „Spadic-Control“ in der mittlerweile veralteten Python-Version 2.7 geschrieben wurde, war es notwendig den Programmcode durch einige Änderungen mit der Python Version 3 kompatibel zu machen. Im Zuge dieser Anpassungen wurde auch die Struktur des Quellcodes modernisiert, wobei noch stärker auf die Prinzipien einer objektorientierten Programmierung geachtet wurde.

Die Controller-Klassen verbinden Frontend und Backend miteinander, sie speichern zur Laufzeit des Programms, in welchem Zustand sich welche Komponente befindet und gleichen dies mit dem Backend ab. Damit dies auch bei Multithreading reibungslos funktioniert, wurde der von Qt bereitgestellte „Signal - Slot“ Mechanismus implementiert. Programm-Objekte senden hierbei sogenannte Signale aus, wenn sie mit anderen Objekten kommunizieren möchten. Diese Signale aktivieren dann in den adressierten Objekten bestimmte Funktionen, bezeichnet als „Slots“.

Um diese Funktion genauer zu erklären, wird in Listing 4.1 ein Auszug aus dem Quellcode der Controller-Klasse für den SPADIC gezeigt. In den Zeilen 3 und 4 werden hier neue Signale definiert, welche die Ausführung des Baseline-Anpassungs-Algorithmus steuern sollen. Der Algorithmus ist im Backend implementiert. In Zeile 13 des Code-Auszuges wird das Signal `do_adjustBaseline`, welches den Algorithmus starten soll, mit einer Funktion namens `adjust_baseline` mittels der Funktion `connect` verbunden. Ab jetzt wird die Funktion `adjust_baseline` immer dann aufgerufen, wenn irgendeine Komponente des Programms das Signal `do_adjustBaseline` emittiert. Hier kann zum Beispiel ein Button in der GUI das Signal emittieren, wenn er angeklickt wurde. Die Funktion ist ab Zeile 16 dargestellt. Der Decorator `@pyqtSlot(int)` definiert, dass diese Funktion ein Slot ist, der von Signalen gesteuert werden kann. Die Slot-Funktion emittiert in Zeile 26 selbst auch ein Signal, das in diesem Fall anderen Programmkomponenten signalisiert, dass der Baseline-Anpassungs-Algorithmus erfolgreich beendet wurde.

Der Programmablauf im Beispiel wird durch Abbildung 4.2 vereinfacht dargestellt. Die Signale bilden eine Vermittlungsebene zwischen dem Frontend

```
1  class SpadicSignals(QObject):
2    # Definiere benötigte Signale:
3    do_adjustBaseline = pyqtSignal(int)          # Signal 1
4    done_adjustBaseline = pyqtSignal(list,list)  # Signal 2
5
6  class SpadicController(QObject):
7    def __init__(self, _num_spadics = 1):
8      super().__init__()
9      # Lade Signale aus der zuvor definierten Klasse:
10     self.signals = SpadicSignals.getSpadicSignals()
11     ...
12     # Verbinde Signal 1 mit einer Funktion (Slot):
13     self.signals.do_adjustBaseline.connect(self.adjust_baseline)
14     ...
15
16   @pyqtSlot(int)
17   def adjust_baseline(self, baseline_val: int):
18     try:
19       # Führe eine Funktion im Backend aus:
20       baseline_res = self.spadics[0].baseline_adjust(baseline_val
             , None, self.bsl_adj_progress)
21     except Exception:
22       LOG.error("adjust_baseline failed", exc_info=1)
23       self.signals.done_adjustBaseline.emit([],[])
24     x_res, y_res = self.dict_to_lists(baseline_res)
25     # Emittiere Signal 2:
26     self.signals.done_adjustBaseline.emit(x_res,y_res)
```

Listing 4.1: Python Code Auszug aus der Controller Klasse für den SPADIC. Dieses Beispiel soll die Verwendung des Signal-Slot Mechanismus zeigen.

und der Controller-Ebene.

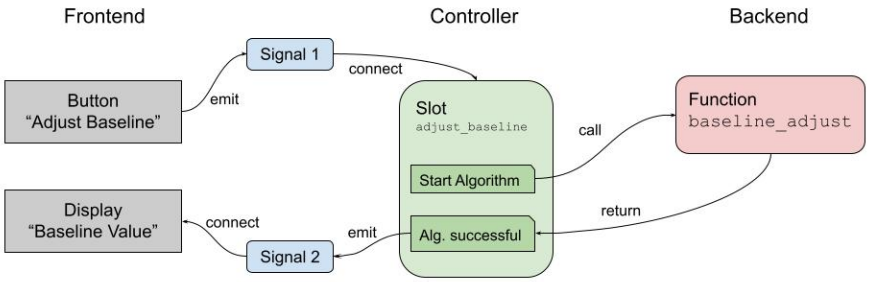

Abbildung 4.2.: Schematische Darstellung des Signal-Slot-Mechanismus am Beispiel der Funktion `adjust_baseline`.

4.3. Das Spadic-Chiptest Programm

Dieser Abschnitt erklärt die Funktionen von Spadic-Chiptest anhand der grafischen Benutzeroberfläche.

Um das Programm zu starten, muss der Root-Ordner des Programms in einer bash-Shell geöffnet werden. Darin wird das Main-Script mit Python 3 gestartet, der Befehl dafür lautet in der Regel `python main.py`. Daraufhin öffnet sich das Hauptfenster der GUI, Abbildung 4.3 zeigt einen Screenshot davon. Das Fenster gliedert sich in drei Hauptbereiche. Auf der linken Seite befinden sich drei Buttons mit zugehörigen Kontrollleuchten, die den Status der jeweiligen Komponente anzeigen, in Abbildung 4.3 markiert mit einer roten 1. Rechts davon, markiert mit einer blauen 2, befindet sich der Tab-Bereich, hier werden, je nach gewähltem Reiter, verschiedene Informationen, Einstellungen und Messwerte angezeigt. Die Tabs werden später einzeln genauer beschrieben.

Im unteren Bereich des Hauptfensters, im Screenshot markiert mit einer grünen 3, befindet sich das **Log**, hier ist die Kommandozeilen-Ausgabe des Programms zu sehen, je nach gewähltem Log-Level ist diese mehr oder weniger ausführlich. Die verfügbaren Log-Level sind `Debug, Info, Warning, Error, Critical`; wobei Debug alle verfügbaren Meldungen anzeigt und Critical nur solche, die durch einen Absturz des Programms erzeugt wurden. Alle Log-Ausgaben werden auch in einer Log-Datei gespeichert, hier ist das Log-

Level, unabhängig von dem in der GUI gewählten, standarmäßig auf `Debug` eingestellt, sodass alle Informationen in der Datei zu finden sind.

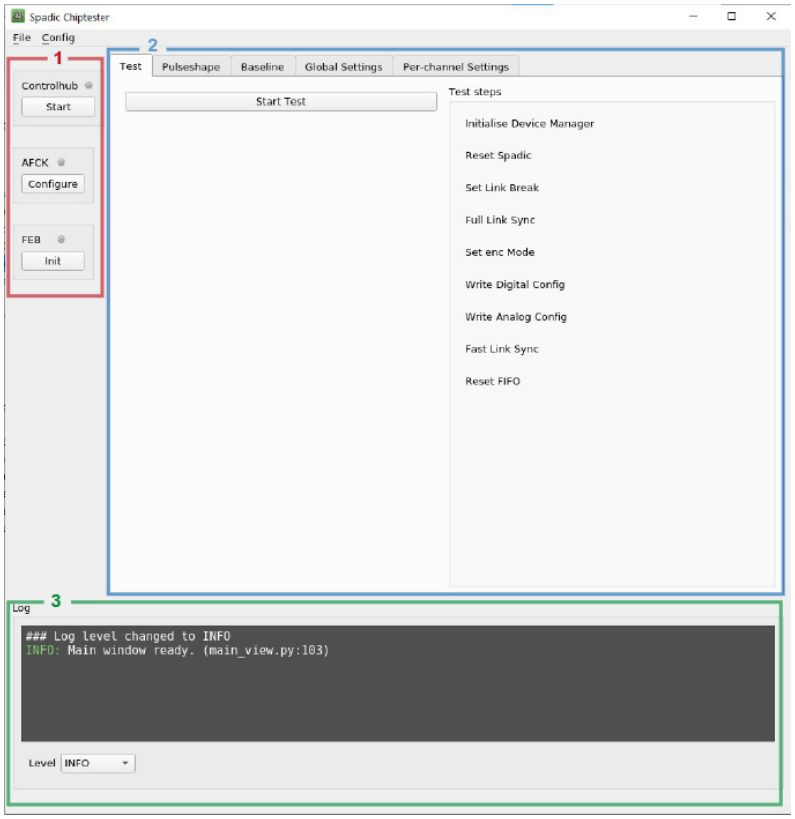

Abbildung 4.3.: Das Hauptfenster von Spadic-Chiptest nach dem Starten der Anwendung.

Die linke Seitenleiste zeigt den Status von Controlhub, AFCK und FEB, und sie bietet jeweils eine Aktion dazu an, siehe Abbildung 4.4. Die Status-Kontrollleuchten sind nach dem Programmstart zunächst grau, dies bedeutet, der Status ist unbekannt. Wenn die Komponente ordnungsgemäß funktioniert ist die Kontrollleuchte grün, bei einem Fehler rot.

Durch Click auf Start unter „Controlhub“ wird der Controlhub-Daemon der IPbus-Software gestartet. Dieser wird benötigt, um das AFCK via IPbus-

Protokoll zu erreichen. Falls Controlhub bereits läuft, passiert nichts. Es wird lediglich eine entsprechende Meldung im Log angezeigt. Alle Optionen des Controlhubs sind in der Menüleiste unter Config → Controlhub verfügbar.

Der Button Configure unter „AFCK“ startet die Übertragung der Firmware auf das AFCK. Dieser Vorgang kann einige Sekunden dauern. Die Scripte, die zum Steuern des JTAG-Programmers verwendet werden und das Firmware-Bitfile können in einem separaten Menü verändert werden. Hierzu wird in der Menüleiste Config → AFCK ausgewählt.

Der Button Init unter „FEB“, startet die Initialisierungssequenz für das angeschlossene FEB. Dieser Vorgang kann einige Sekunden dauern. Diese Funktion ist für das direkte Initialisieren eines bereits getesteten FEBs gedacht, um danach andere Aufgaben auszuführen.

Abbildung 4.4.: Die linke Seitenleiste. Controlhub und AFCK sind bereit, ein Test kann gestartet werden.

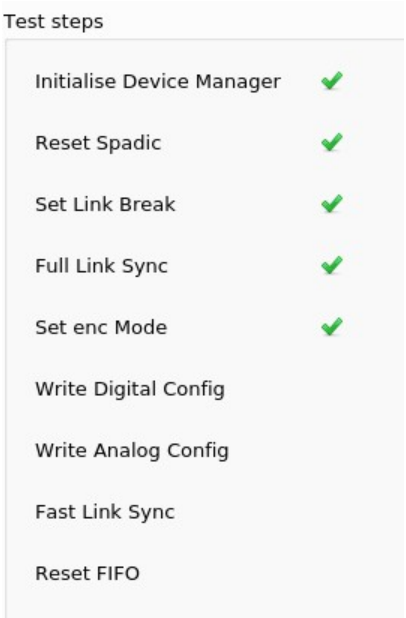

Abbildung 4.5.: Die Anzeige der Testschritte während des automatischen Tests.

Der Test Tab zeigt den automatisch ausgeführten Test. Bevor der Test gestartet werden kann, muss sichergestellt sein, dass der Controlhub läuft und dass das AFCK konfiguriert ist. Dies ist in der linken Seitenleiste erkennbar: Wenn die Kontrollleuchten der beiden oberen Komponenten grün sind, kann der Test gestartet werden, ansonsten ist zuerst Start beziehungsweise Configure anzuklicken. Der Start-Test Button startet dann den automatischen Test. In der aktuellen Version wird hierbei eine Initialisierungssequenz für den SPADIC Schritt für Schritt ausgeführt. Durch grüne Haken oder rote Kreuze wird angezeigt, ob der jeweilige Schritt erfolgreich beendet wurde. Wenn alle Schritte erfolgreich beendet wurden, gilt der Test als bestanden. Dies wird in Abbildung 4.5 dargestellt, der gesamte Tab ist in Abbildung 4.3 zu sehen. Die Befehlszeilen-Ausgabe des Programms während des Tests wird in der Log-Datei abgespeichert. Diese befindet sich im Unterordner `log/`.

Während der Testsequenz werden verschiedene Register des SPADIC angesteuert und die Antwort des Chips überprüft. Ein Hauptpunkt ist die Synchronisierung des e-Links zwischen FEB und AFCK. Ein weiterer wichtiger Test ist das Schreiben der Schieberegister mit den Einstellungen für den digitalen und analogen Teil des Chips. Die Funktionalität oder Qualität der Datenmessung wird zurzeit nicht automatisch überprüft, es ist aber möglich, nach Abschluss des Tests entweder im Pulseshape Tab oder im Baseline Tab eine Datenmessung auszuführen, um die Funktionalität zu überprüfen.

Der Pulseshape Tab ermöglicht es, einzelne gemessene Signale von ausgewählten SPADIC-Kanälen anzuzeigen. Er besteht aus drei Buttons und einem matplotlib-Widget, in welchem die Messdaten grafisch dargestellt werden. Das Widget verfügt auch über eine eigene Werkzeugleiste, die es erlaubt, die Ansicht zu verändern oder abzuspeichern. Show Pulseshape zeigt den theoretischen Signalverlauf als Kurve im Plot. Get Data veranlasst den SPADIC, mit jedem Kanal eine Messung durchzuführen, er wird dazu einmal extern getriggert. Danach kann mit Select Channels ein Dropdwon-Menü geöffnet werden, in welchem beliebige Kanäle zur Ansicht ausgewählt werden können. Abbildung 4.6 zeigt den Tab mit zwei ausgewählten Kanälen. Der Signalverlauf ist hier flach, weil der SPADIC nicht an einem Detektor angeschlossen ist und somit nur elektronisches Rauschen aufgezeichnet wird.

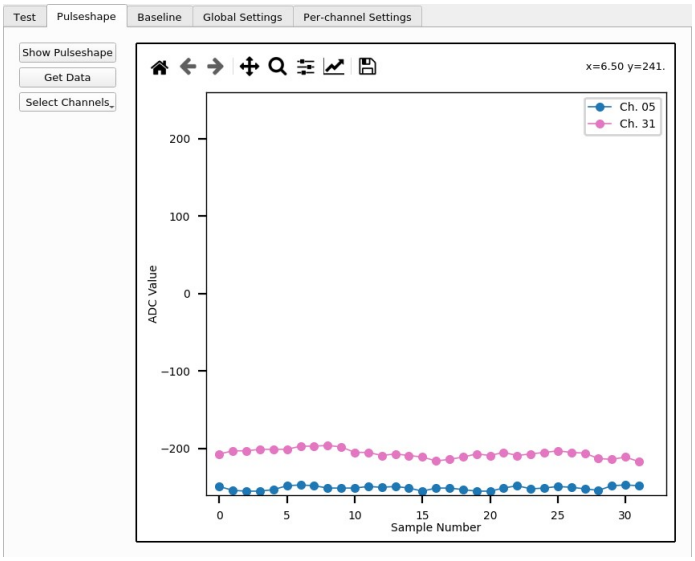

Abbildung 4.6.: Der Pulseshape Tab mit Signalen von zwei Kanälen.

Der Baseline Tab erlaubt dem User die Anzeige und Anpasung der Baseline. Er ist ähnlich aufgebaut wie der Pulseshape Tab. Die Baseline ist das Nullsignal, ein von null verschiedener Messwert, obwohl kein Signal vorliegt. Sie wird hauptsächlich durch elektronisches Rauschen verursacht, daher ist es unmöglich, sie bei einer Messung auszuschließen. Die genaue Kenntnis der Baseline ist daher wichtig, um später das gemessene Signal entsprechend korrigieren zu können. Die Korrektur wird einfacher, wenn die Baseline in allen Messkanälen ungefähr gleich ist. Deshalb bietet der SPADIC eine Möglichkeit, die Vorverstärkung für jeden Kanal einzeln anzupassen und so die Baseline für alle Kanäle auf den gleichen Wert zu bringen.

Um die Baseline mit den aktuellen Einstellungen zu messen, wird auf Get Baseline geklickt. Der SPADIC wird dann zehnmal getriggert, um für jeden Kanal einige Hit-Messages zu erhalten. Die Baseline einer einzelnen Hit-Message wird berechnet als Mittelwert der bis zu 32 ADC-Werte, die den zeitlichen Signalverlauf darstellen. Um die Baseline eines Kanals zu bestimmen, wird über alle verfügbaren Hit-Message-Baselines gemittelt. Die Anzahl der Hit-Messages pro Kanal beträgt bei zehn Triggern ebenfalls in der Regel zehn; in seltenen Fällen fehlen einzelne Hit-Messages. Die so erhaltenen Werte

werden dann als Histogramm geplottet, das für jeden Kanal einen Eintrag bei dem ermittelten Baseline-Wert hat. Ein globaler Mittelwert über alle Kanäle wird ebenfalls berechnet und im Log ausgegeben.

Um die Baseline nun an einen gewählten Wert anzupassen, wird dieser Wert in der Eingabemaske eingegeben. Realistische Werte liegen zwischen −200 und −250. Durch Klicken auf Adjust Baseline wird die Anpassung gestartet. Dieser Vorgang benötigt mehrere Minuten, er kann mit einem Klick auf Cancel abgebrochen werden. Während der Anpassung wird nach jeder Iteration des Algorithmus das Histogramm aktualisiert. Wenn die Baseline eines Kanals sich während der Anpassung nicht mehr weiter dem eingestellten Wert annähert, wird der Kanal für die weiteren Iterationen ignoriert. Die Anpassung endet, wenn alle Kanäle den Sollwert erreicht haben oder ignoriert wurden, so wird eine Endlosschleife verhindert. Abbildung 4.7 zeigt die Baseline aller Kanäle ohne Anpassung und Abbildung 4.8 nach der Anpassung auf den Wert −220. Es ist zu erkennen, dass fast alle Kanäle genau den eingestellten Wert erreichen. Die Kanäle, die den Wert nicht ganz erreichen, schöpfen in der Beispielmessung bereits alle Möglichkeiten für die Anpassung aus. Es wird vermutet, dass hier die Referenzwiderstände minimal vom Sollwert abweichen. Es ist allerdings auch zu bedenken, dass der Chip nicht an einem Detektor angeschlossen war, wodurch dessen Kapazität die Messeingänge nicht wie berechnet beeinflusst. Lediglich Kanal 3 reagiert in diesem Fall gar nicht auf die Anpassung, hier wird von einem Defekt im Chip oder auf der Platine ausgegangen.

Der Global Settings Tab beinhaltet die wichtigsten Einstellungen des SPADIC, die nicht für einzelne Kanäle gelten. Um die aktuellen Einstellungen anzuzeigen, müssen sie zuerst vom SPADIC eingelesen werden, dazu wird auf Read geklickt. Nun ist es möglich, Einstellungen zu verändern. Damit die Änderungen wirksam werden, müssen die Einstellungen wieder auf den SPADIC geschrieben werden. Dies erfolgt mit einem Klick auf Write. Es ist auch möglich, die Einstellungen für spätere Verwendung in zwei Dateien abzuspeichern. Es werden zwei Dateien angelegt, getrennt nach dem Register für die analogen Einstellungen und dem Register für die digitalen. Die Dateien verwenden das menschenlesbare `json`-Format. Save speichert die aktuellen Einstellungen im ausgewählten Ordner ab. Es werden immer dieselben Dateinamen verwendet, `analog_settings.json` und `digital_settings.json`. Es ist auch möglich, Einstellungen in diesen Dateien zu verändern. Dazu wird die gewünschte Än-

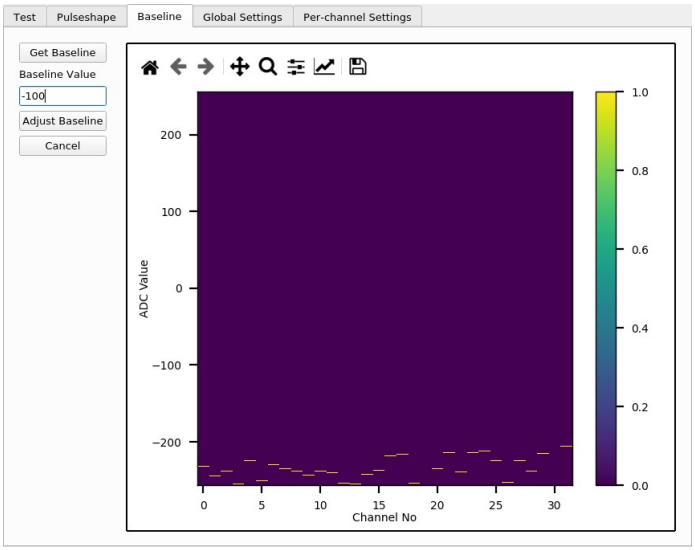

Abbildung 4.7.: Der Baseline Tab vor der Anpassung der Baseline.

derung in einem Text-Editor durchgeführt, die Datei abgespeichert und dann mit einem Klick auf Load in das Programm geladen. Anschließend muss die Änderung wieder auf den SPADIC übertragen werden.

Die Einstellungen, welche im Programm bereits implementiert wurden, werden nun noch näher erklärt. Vollständige Listen aller Einstellungen befinden sich in Anhang A.1 und A.2, getrennt nach Einstellungen für Analog- und Digitalteil.

REG userpin: Die Single-FEBs der SPADIC Version 2.2 haben zwei Leuchtdioden, die durch den SPADIC gesteuert werden. Hier können diese ein- oder ausgeschaltet werden.

REG select mask: Lässt den Benutzer auswählen, welche der 32 Zeit-Samples in den Hit-Messages ausgegeben werden sollen. Zum Beispiel sorgt eine `1` an der dritten Position dafür, dass das dritte Zeit-Sample aktiviert ist. Mit einer `0` wird das entsprechende Sample deaktiviert.

REG use average baseline: Der SPADIC verfügt über ein Feature, das den Durchschnittswert der Baseline in das erste Zeit-Sample der Hit-Messages schreibt.

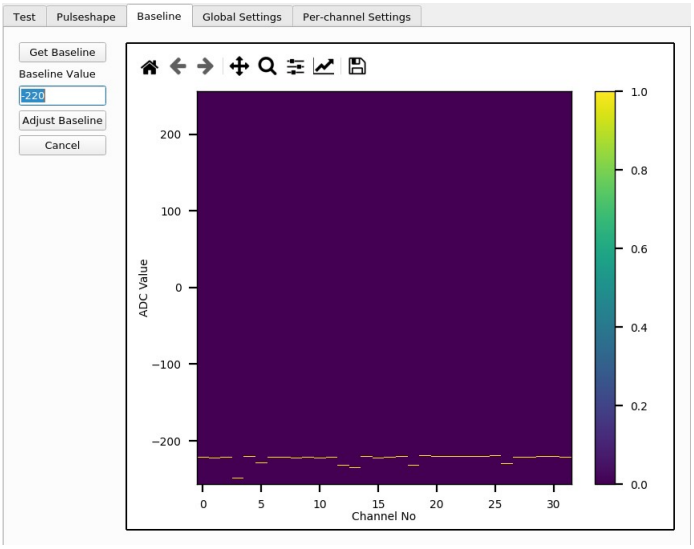

Abbildung 4.8.: Der Baseline Tab nachdem die Baseline auf den Wert −220 angepasst wurde.

REG disable channels: 32 Bit für 32 Kanäle, bei einer `0` ist der Kanal nicht deaktiviert, bei einer `1` wird er deaktiviert.

REG disable epoch: Wenn der 8 Bit Zeitstempel des SPADIC überläuft wird eine „epoch message“ gesendet. Mit dieser Einstellung werden diese Nachrichten unterdrückt, getrennt für die beiden Chip-Hälften.

REG threshold: Ermöglicht das gleichzeitige setzen von Trigger-Schwellen für alle Kanäle. Hier können Werte zwischen −255 und 255 eingegeben werden.

Baseline Trim N: Verändert die Lage der Baseline für alle Kanäle. Wertebereich von 0 bis 127. Bei kleineren Werten liegt die Baseline für alle Kanäle höher.

Der Per-Channel Settings Tab beinhaltet Einstellungen, die für die Kanäle des SPADIC einzeln veränderbar sind. Er ist daher als Liste aus 32 Kanälen aufgebaut, welche die folgenden Spalten enthält:

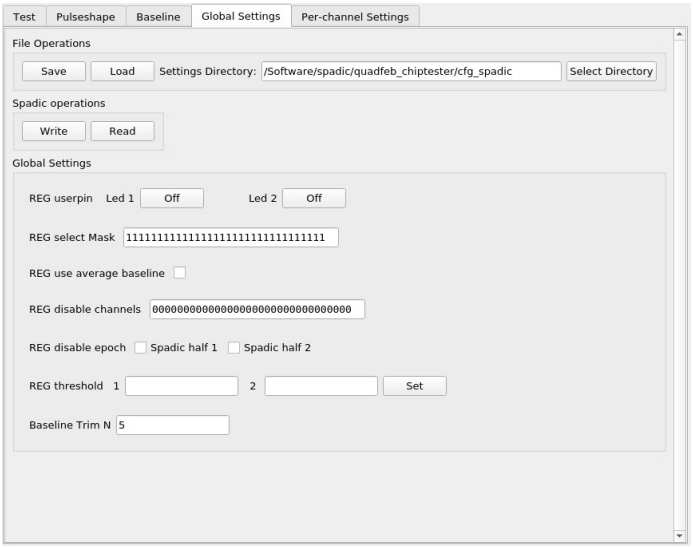

Abbildung 4.9.: Der Global Settings Tab. Oben sind Steuerungselemente, um Einstellungen zu speichern und zu übertragen. Darunter befinden sich die Einstellungen.

Enabled: Legt fest, ob der Kanal im Chip aktiviert ist. Deaktivierte Kanäle werden nicht ausgelesen. Korrespondiert mit REG disable channels.

Baseline Trim P: Verändert die Lage der Baseline für diesen Kanal. Wertebereich von 0 bis 127. Größere Werte heben die Baseline stärker an.

Threshold 1: Schwelle, ab der die Hit-Erkennungs-Logik das Signal als Hit interpretiert. Siehe Kapitel 3.2.1. Wertebereich zwischen -255 und 255.

Threshold 2: Wie Threshold 1.

Baseline: Der zuletzt gemessene Baseline-Wert für diesen Kanal.

Um hier getätigte Änderungen wirksam zu machen, oder um sie abzuspeichern, werden die Steuerelemente im Global Settings Tab verwendet. Hier wird nichts angezeigt, wenn die Einstellungen noch nicht mit Read oder Load in das Programm geladen wurden.

5. Zusammenfassung

Das Resultat dieser Arbeit ist ein funktionstüchtiges Programm, das die gestellten Anforderungen erfüllt. Anhand dieser Anleitung ist es auch unerfahrenen Nutzern möglich, einen SPADIC im Labor zu konfigurieren und zu testen.

Zum Erreichen dieses Ziels war es zunächst wichtig, das Verhalten des SPADIC 2.2 und die verfügbaren Einstellmöglichkeiten zu verstehen. Da bisher keine offizielle Dokumentation des SPADIC existiert, wurden die relevanten Informationen zusammengetragen und in dieser Arbeit dargestellt. Als Nächstes wurde ein Protoyp-Setup für die Steuerung des SPADIC anhand eines bestehenden Konzeptes mit dem AFCK-Board im Labor aufgebaut und mit einer bereits vorhandenen Software in Betrieb genommen. Nachdem die Software analysiert und verstanden war, konnten die weiteren Schritte zur Erstellung eines neuen Programms geplant werden. Die vorhandene Implementierung des IPbus-Protokolls wurde überarbeitet und für die Verwendung mit einer grafischen Benutzeroberfläche vorbereitet. Es erfolgte eine umfangreiche Einarbeitung in das Qt-Framework, mit welchem dann die GUI aufgebaut und mit dem Backend verbunden wurde. Abschließend wurden die Funktionen des neuen Programms getestet.

5.1. Ausblick

Zum aktuellen Zeitpunkt ist ein konkreter Anwendungsfall für das Programm noch nicht gegeben, da die zu testende Elektronik noch nicht zur Verfügung steht. Die hier betrachtete Version 2.2 des SPADIC wird nicht die finale Version sein, die im CBM Experiment eingesetzt wird. Sobald die Spezifikationen der finalen Version feststehen, wird es möglich sein, eine detaillierte Testprozedur zu entwickeln. Anhand dieser Prozedur kann dann ein physischer Teststand im Labor aufgebaut werden, der es ermöglicht, viele FEBs effizient zu testen. Die Software muss dann entsprechend angepasst und erweitert werden, um unter anderem den Teststand steuern zu können. Zusätzlich könnte es sinnvoll sein, auch eine Evaluation der Messqualität aller SPADIC Kanäle durchzuführen, indem ein Pulser-Signal in die Messeingänge des Chips eingespeist und mit dem Messergebnis verglichen wird.

6. Verzeichnisse

Abkürzungen

ADC Analog to Digital Converter.

AFCK AMC FMC Carrier Kinter.

AMC Advanced Mezzanine Card.

ASIC Application-Specific Integrated Circuit.

CBM Compressed Baryonic Matter.

CERN Conseil Européen pour la Recherche Nucléaire.

CRI Common Readout Interface.

DAQ Data Acquisition.

FAIR Facility for Antiproton and Ion Research.

FEB Front-End-Board.

FEE Front-End-Elektronik.

FMC FPGA Mezzanine Card.

FPGA Field-Programmable Gate Array.

GBTx GigaBit Transceiver ASIC.

GEM Gas Electron Multiplier.

GSI Gesellschaft für SchwerIonenforschung.

GSI GSI - Helmholtzzentrum für Schwerionenforschung.

GUI Graphical User Interface.

HAL Hardware Access Library.

IKF Institut für Kernphysik Frankfurt.

LQCD Lattice-QCD.

MUCH Muon Chamber.

MVD Micro Vertex Detector.

MWPC Multi Wire Proportional Chamber.

PSD Projectile Spectator Detector.

QCD Quantenchromodynamik.

QED Quantenelektrodynamik.

QGP Quark-Gluon-Plasma.

RICH Ring Imaging Cherenkov Detector.

ROB ReadOut Board.

SIS Schwer-Ionen Synchrotron.

SPADIC Self-triggered Pulse Amplification and Digitization asIC.

STS Silicon Tracking System.

TOF Time Of Flight.

TRD Transition Radiation Detector.

Quellen

[1] Helmut Satz. „The Thermodynamics of Quarks and Gluons“. In: *The Physics of the Quark-Gluon Plasma: Introductory Lectures*. Hrsg. von Sourav Sarkar. Bd. 785. Lecture Notes in Physics. Springer Berlin, Heidelberg, 2010. DOI: 10.1007/978-3-642-02286-9.

[2] J. Bartke. *Introduction to Relativistic Heavy Ion Physics*. World Scientific, 2008. DOI: 10.1142/1881.

[3] Gordon Baym u. a. „From hadrons to quarks in neutron stars: a review“. In: *Rept. Prog. Phys.* 81.5 (2018), S. 056902. DOI: 10.1088/1361-6633/aaae14. arXiv: 1707.04966 [astro-ph.HE].

[4] *The Transition Radiation Detector of the CBM Experiment at FAIR: Technical Design Report for the CBM Transition Radiation Detector (TRD)*. Techn. Ber. FAIR Technical Design Report. Darmstadt, 2018, 165 p. DOI: 10.15120/GSI-2018-01091. URL: https://repository.gsi.de/record/217478.

[5] Patrick Dahm. *Setups & challenges of the construction of CBM*. Presentation, 40th CBM Collaboration Meeting. GSI Helmoltzzentrum für Schwerionenforschung, Okt. 2022.

[6] Philipp Kähler. *TRD production status*. Presentation, 40th CBM Collaboration Meeting. Westfälische Wilhelms-Universität Münster, Institut für Kernphysik, Okt. 2022.

[7] Peter Fischer. „Spadic 2.2 Chip Manual“. Version 1.2. Institut für Technische Informatik der Universität Heidelberg, Okt. 2022. Work in Progress.

[8] *Technical Design Report for the CBM Online Systems – Part I*. Techn. Ber. manuscript submitted for publication, FAIR Technical Design Report. Darmstadt, 2022, 202 p. URL: https://git.cbm.gsi.de/doc/tdr-online/-/jobs/artifacts/master/raw/online_part1.pdf?job=build.

[9] David Schmidt. „Firmware Development for the TRD Data Processing Board Prototype“. Masterthesis. Institut für Kernphysik Frankfurt, Nov. 2019.

[10] C. Ghabrous Larrea u. a. „IPbus: a flexible Ethernet-based control system for xTCA hardware“. In: *Journal of Instrumentation* 10.02 (Feb. 2015), S. C02019. DOI: 10.1088/1748-0221/10/02/C02019. URL: https://dx.doi.org/10.1088/1748-0221/10/02/C02019.

Anhang A.

SPADIC Konfiguration

A.1. Einstellungen des Analogteils

Das Schieberegister für den Analogteil besteht aus 584 Bit, sie setzen sich zusammen aus 136 Bit für Einstellungen, die den ganzen Chip betreffen und $32 \cdot 14 = 448$ Bit für Einstellungen, die für jeden Kanal separat gesetzt werden können.

Analog-Register, globale Einstellungen

Bit Pos.	**Name**	**Beschreibung**
0	*unused*	
01 ... 07	VNDel	ADC
09 ... 15	VPDel	ADC
17 ... 23	VPLoadFB2	ADC
25 ... 31	VPLoadFB	ADC
33 ... 39	VPFB	ADC
41 ... 47	VPAmp	ADC
49 ... 55	baselineTrimN	ADC, Trimmt Baseline nach unten
56	DecSelectNP	CSA
57 ... 63	pCascP	CSA
64	SelMonitor	CSA
65 ... 71	nCascP	CSA
73 ... 79	pSourceBiasN	CSA
81 ... 87	pSourceBiasP	CSA
89 ... 95	nSourceBiasN	CSA
97 ... 103	nSourceBiasP	CSA
105 ... 111	pFBN	CSA
113 ... 119	nFBP	CSA
121 ... 127	pCascN	CSA
128 ... 135	nCascN	CSA

Analog-Register, Einstellungen pro Kanal

Bit Pos.	**Name**	**Beschreibung**
137 ... 143	baselineTrimP_0	Kanal 0, Trimmt Baseline nach oben.
144	enSignalAdc_0	Kanal 0
145	enMonitorAdc_0	Kanal 0
146	ampToBus_0	Kanal 0
147	enHalfGain_0	Kanal 0, halbiert Ausgang des Vorverstärkers
148	*unused*	Kanal 0
149	enCRRC2_N_0	Kanal 0, Setze auf `0` um Second-Order-Shaper zu aktivieren.
151 ... 157	baselineTrimP_1	Kanal 1
⋮	⋮	⋮
571 ... 577	baselineTrimP_31	Kanal 31, Trimmt Baseline nach oben.
578	enSignalAdc_31	Kanal 31
579	enMonitorAdc_31	Kanal 31
580	ampToBus_31	Kanal 31
581	enHalfGain_31	Kanal 31, halbiert Ausgang des Vorverstärkers
582	*unused*	Kanal 31
583	enCRRC2_N_31	Kanal 31, Setze auf `0` um Second-Order-Shaper zu aktivieren.

A.2. Einstellungen des Digitalteils

Das Schieberegister für den Digitalteil verwendet 241 Speicheradressen unterschiedlicher Länge, die maximale Anzahl Bits pro Adresse ist 15. Dazu kommen noch zwei Befehlsadressen am Ende.

Digital-Register

Adresse	**Name**	**N Bits**	**Beschreibung**
1	REG_userpin	2	Set Userpin
2	REG_readoutEnabled	1	
3	REG_compDiffMode	1	`0` Direkter oder `1` differentieller Trigger-Modus
4	REG_hitWindowLength	6	Anzahl von Samples innerhalb derer Retrigger als Mutlihit behandelt wird.
5	REG_selectMask_0	15	Welche der 32 Samples
6	REG_selectMask_1	15	in die Hit-Messages
7	REG_selectMask_2	2	geschrieben werden.
8	REG_selectMask2_0	15	Wie vorherig,
9	REG_selectMask2_1	15	aber für Mutlihits.
10	REG_selectMask2_2	2	
11	REG_useAverageBaseline	1	Aktiviere laufende Baseline Mittelwert-Bildung.
12	REG_bypassFilterStage	5	DSP, `0x1f` = alle Filterstufen deaktiviert.
13	REG_aCoeffFilter_0	15	DSP, 3·6 Bit Koeffizienten
14	REG_aCoeffFilter_1	3	DSP, für Stufen 1-3
15	REG_bCoeffFilter_0	15	DSP, 4·6 Bit Koeffizienten
16	REG_bCoeffFilter_1	9	DSP, für Stufen 0-3
17	REG_scalingFilter	9	DSP, output Scaling
18	REG_offsetFilter	9	DSP, output Offset

Digital-Register, Fortsetzung

Adresse	**Name**	**N Bits**	**Beschreibung**
19...56	REG_neighborSelectMatrixA	$38 \cdot 15$	Neighbor Trigger Matrix für
57	REG_neighborSelectMatrixA_38	6	Chip-Hälfte A
58...95	REG_neighborSelectMatrixB	$38 \cdot 15$	Neighbor Trigger Matrix für
96	REG_neighborSelectMatrixB_38	6	Chip-Hälfte B
97	REG_disableChannelA_0	15	Deaktiviere Kanäle
98	REG_disableChannelA_1	1	
99	REG_disableChannelB_0	15	wie 97
100	REG_disableChannelB_1	1	
101	REG_disableEpochChannelA	1	Deaktiviere Epoch Message
102	REG_disableEpochChannelB	1	
103	REG_enableTestOutput	1	
104	REG_testOutputSelGroup	1	
105	REG_enableTestInput	1	
106	REG_enableAdcDec_0	15	
107	REG_enableAdcDec_1	6	
108	REG_triggerMaskA_0	15	
109	REG_triggerMaskA_1	1	
110	REG_triggerMaskB_0	15	
111	REG_triggerMaskB_1	1	
112	REG_analogTrigger	1	
113	REG_enableTriggerOutput	1	
114	REG_softReset	1	
115	REG_analogPowerDownToggle	1	
116	REG_triggerHoldoff	6	Anzahl Samples in denen nicht erneut getriggert wird.

Digital-Register, Fortsetzung

Adresse	**Name**	**N Bits**	**Beschreibung**
117	REG_threshold1_0	15	Trigger Threshold 1
⋮	⋮	⋮	für alle Kanäle,
136	REG_threshold1_19	3	$32 \cdot 9 = 19 \cdot 15 + 3$ Bit
137	REG_threshold2_0	15	Threshold 2
⋮	⋮	⋮	
156	REG_threshold2_19	3	
157	REG_threshold3_0	15	Threshold 3
⋮	⋮	⋮	
176	REG_threshold3_19	3	
177	REG_threshold4_0	15	Threshold 4
⋮	⋮	⋮	
196	REG_threshold4_19	3	
197	REG_threshold5_0	15	Threshold 5
⋮	⋮	⋮	
216	REG_threshold5_19	3	
217	REG_threshMuxA_0	15	Multiplexer A Zuordnung
⋮	⋮	⋮	für alle Kanäle
223	REG_threshMuxA_6	6	$32 \cdot 3 = 6 \cdot 15 + 6$ Bit
224	REG_threshMuxB_0	15	Multiplexer B
⋮	⋮	⋮	
230	REG_threshMuxB_6	6	
231	REG_threshMuxC_0	15	Multiplexer C
⋮	⋮	⋮	
237	REG_threshMuxC_6	6	
238	REG_hitDetLut_0	15	Hit-Detection-
239	REG_hitDetLut_1	15	Look-Up-Table
240	REG_hitDetLut_2	2	
241	REG_useOldHitDet	1	Verwende alte Hit-Detection-Logik
242	CMD_trigger	2	
243	CMD_sync	1	

A.3. Die Neighbor Trigger Matrix

Hier werden weitere Details zu dem in Kapitel 3.2.2 behandelten Thema aufgeführt. Die Neighbor-Trigger-Logik wird für jede Kanalgruppe (Chip-Hälfte)

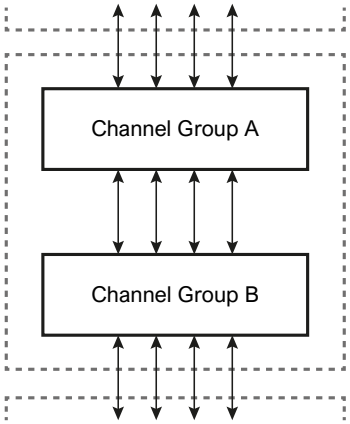

Abbildung A.1.: Die Neighbor Trigger Busse zwischen den Kanalgruppen der SPADICs werden durch die Pfeile symbolisiert.

durch die Neighbor-Trigger-Matrix konfiguriert. Abbildung A.2 zeigt die Matrix mit einer Standardkonfiguration. Das zentrale Quadrat konfiguriert dabei die Trigger zwischen Kanälen der Kanalgruppe. Die außenliegenden Rechtecke stellen die Eingangs- und Ausgangskanäle der Inter-Chip-Busse dar. In jedes Feld kann eine `1` oder eine `0` eingetragen werden. Eine Eins bedeutet: Ein Hit im Quell-Kanal (horizontale Achse) erzeugt einen Neighbor-Trigger im Ziel-Kanal (vertikale Achse). Die Nullen wurden hier der Übersichtlichkeit halber weggelassen. Der Ausgang von „Bus Down“ in Kanalgruppe A ist mit dem Eingang von „Bus Up“ in Kanalgruppe B verbunden.

Um die Konfiguration in das Schieberegister des SPADIC zu schreiben, werden auch alle ausgegrauten Felder mit Nullen gefüllt, wodurch ein Quadrat mit 24×24 Bit entsteht. Dieses Quadrat wird Zeilenweise von links oben nach rechts unten in 15-Bit Stücke zerlegt, die dann in das Schieberegister geschrieben werden. Bit-Anzahl: $24 \cdot 24 = 576 = 38 \cdot 15 + 6$. Registeradressen $19 - 57$ Für Gruppe A und $58 - 96$ für Gruppe B (siehe Tabelle in Anhang A.2).

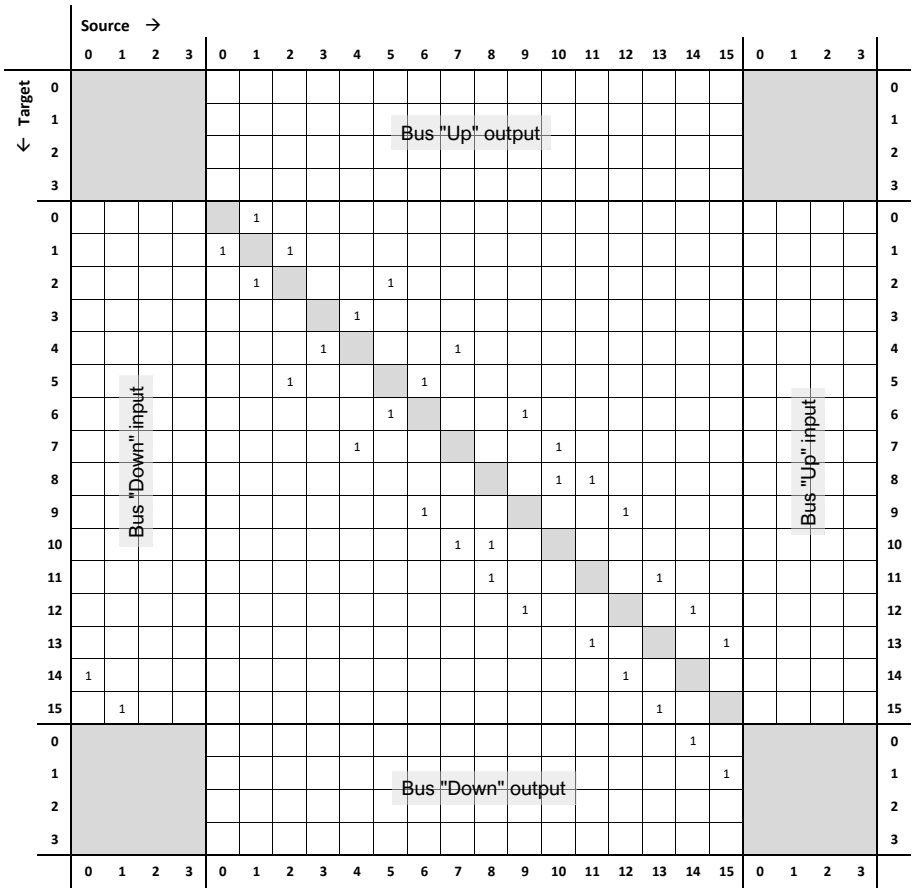

Abbildung A.2.: Die Neighbor Trigger Matrix für eine Kanalgruppe mit 16 Kanälen. „Bus Up“ und „Down“ entsprechen den Pfeilen, die in Abb. A.1 von oben bzw. von unten die Kanalgruppen verbinden.

Anhang B.

Spadic-Chiptest Zusatzmaterial

Hier werden zusätzliche Informationen zur Spadic-Chiptest Software bereitgestellt. Sie ist lizenziert durch die GNU GPL v3, der Quellcode ist verfügbar unter `https://git.cbm.gsi.de/d.spicker/quadfeb_chiptester`. Die korrekte Funktion der Software ist aktuell nur auf pcikf25 gewährleistet, da hier alle Abhängigkeiten installiert sind. Alle Anleitungen und Beschreibungen in dieser Arbeit beziehen sich auf Version `v0.0.1` von Spadic-Chiptest. Die Versionen der verwendeten Programme und Pakete werden in Tabelle B.1 zusammengefasst.

Paket	Version
Python	3.7.3
IPbus	2.8.1
PyQT	5.15.2
Qt	5.11.3
matplotlib	3.4.3

Tabelle B.1.: Im Projekt verwendete Software-Versionen

Die Ordnerstruktur des Programms ist folgendermaßen aufgebaut:

```
/spadic_chiptest/
│
├─ cfg_afck/
│   ├─ AFCK Firmware Bitfile
│   └─ Adress-Tabellen für IPbus Protokoll
├─ cfg_spadic/
│   ├─ Adress-Tabelle für IPbus Protokoll
│   └─ Konfigurationsdateien für die Schieberegister
├─ controllers/
│   └─ Qt-Controller-Klassen
├─ lib_ipbus/
│   └─ Backend: Spadic, IPbus
├─ lib_util/
│   └─ Hilfsklassen
├─ log/
│   └─ Log Dateien
├─ resources/
│   └─ Zusätzlich benötigte externe Resourcen
├─ views/
│   └─ Frontend (GUI): pyQt Klassen und Widgets
│
└─ Lizenz, Readme, main.py
```

Selbstständigkeitserklärung

Erklärung nach §30 (12) Ordnung für den Bachelor- und den Masterstudiengang

Hiermit erkläre ich, dass ich die Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderen fremden Texten entnommen wurden, sind von mir als solche kenntlich gemacht worden. Ferner erkläre ich, dass die Arbeit nicht - auch nicht auszugsweise - für eine andere Prüfung verwendet wurde.

Frankfurt, den 2. Januar 2023

Dennis Spicker